Providing better confidentiality and authentication on the Internet using Namecoin and MinimaLT

Frederic Jacobs
www.fredericjacobs.com
me@fredericjacobs.com

ABSTRACT

In this paper, we introduce a duo of improvements for the Internet that would lead to better security. The authentication model on the Internet is broken and TLS connections have a considerable overhead. We try to address those issues with changes at both the application layer, introducing a replacement for the DNS system, and at the transport layer, a drop-in replacement for TCP built on top of UDP so that it can run on today's internet infrastructure.

1. INTRODUCTION

1.1 Motivation

When the Internet was designed at DARPA, the primary goal was to design a system that could provide interconnection between multiple computers. The Web then came by with the motivation to be able to freely exchange information with anyone. But, over time, people started trusting the internet more and more and started sharing more personal information over it. The Internet was never designed to be ran by so many different entities and the threat model implied that you had to trust the entities running the infrastructure of the internet. Over time, attempts such as TLS and DNSSec were made to secure inherently insecure protocols. In this paper, we don't attempt to fix the current protocols that have a huge overhead but try instead to propose better alternatives.

1.2 Another subsection will come here

2. DOMAIN NAMES AND AUTHENTICITY

Today, if I want to load a page from *www.facebook.com*, my computer will have to first get the Domain Name System record matching that domain. DNS was designed in a hierarchical way and TLD registrations are handled by a single organisation, the ICANN.

So what is wrong with DNS?

When the original Domain Name System was designed, it did not include security; instead it was designed to be a scalable distributed system. DNS requests could thus be spoofed and fake DNS query responses could be served to the clients. Therefore, DNSSEC, a security extension of DNS, attempted to prevent those types of attacks by introducing DNS zone signing. DNS zone signing uses a chain of trust to sign entries hierarchically, this means that the ICANN has a key that is known as the root anchor which is the starting point of a trust chain.

The fact that even today, DNSSEC is having issues being deployed at a larger scale has a lot to do with its complicated design. Getting DNSSEC right is hard and it leads to the centralisation of the internet domain name system because few registrars are deploying DNSSEC and other registrars are still requesting their signing keys. Hierarchical trust structures require us to blindly believe that the root keys are not compromised and bring us back to a feudatory system where we need to beg some lords (here companies) for protection to get the keys.

The ICANN has been taking worrying measures that remove the ability to users to register domains anonymously[**?**]. Our systems shouldn't be designed in such a way that a change in policy would make their security obsolete.

Hence, we want to design a distributed system where anyone can register a domain with a fast and efficient registration process.

But how can we verify authenticity? Even if my domain name system returns the right IP address how do I know for sure that I'm establishing a connection with the client I want. Today, we are using yet another hierarchical system to verify authenticity, namely *SSL certificates*. This means that in addition to trusting ICANN, we will have to trust hundreds of Root Certificate Authorities that are shipped with our browsers.[**?**]

If only one of those 100s of CA gets compromised, it could result in the man-in-the-middling of any website without any warning since a root certification authority can generate a fake valid certificate for any website.[1] Certificate authorities getting compromised happened in the past already. For instance, the Dutch certificate authority DigiNotar got compromised[**?**] and the certificate was used to target Iranian citizens.

[1] There are some restrictions on some certificates in Chrome

Now that we are convinced that the hierarchical trust model of the internet is broken, what measures have already been taken to fix authentication on the Internet?

2.0.1 Certificate Pinning

The Chrome Security team led the way by implementing certificate pinning. Certificate pinning is a effective measure to counter man-in-the-middle attacks for today's internet. Certificate pinning works by shipping *pins*[2] in the browser's binary.[**?**] Every time a user loads a pinned website, the certificate fingerprint is compared to the one provided in the binary. If the fingerprint matches, the client continues the SSL handshake. Otherwise, an error message is shown to the user explaining a secured connection couldn't be established.

Although this is a very efficient method to verify SSL certificates, this method doesn't scale and is hard to maintain at a larger scale. In addition to that, the Chrome team needs to verify the "pin" definition manually before merging every pin request into the code branch. Therefore only larger websites do have certificate pins.

2.0.2 TACK

TACK is a proposal by Moxie Marlinkspike and Trevor Perrin, it's a way to 'pin' TLS servers to the correct public key even when a Certificate Authority is telling you differently. Although this is a promising proposition, it doesn't protect against an attacker that has a long-term MITM capability since pins are set on the first connections and do expire after some time.[**?**]

2.0.3 DANE

The IETF proposal called *DANE* is an attempt at making large scale certificate pinning but by distributing the certificate fingerprint by DNS. This would enable website owners to specify their certificate fingerprint as a DNS entry and visitors would thus be able to verify the authenticity of the server.

Even if we consider that DNSSEC does provide good security, this system does still rely on trusting the Domain Name System and it's hierarchical structure.

2.0.4 Tor Hidden Services

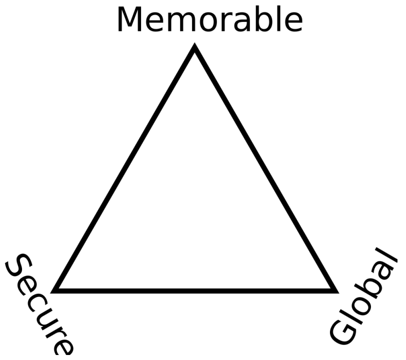
Tor hidden services are reachable by hashes of public keys. This is of course the ideal case when it comes to security because the address itself contains information about the key. Unfortunately, humans are not good at remembering pseudorandom 16-character long strings.

2.1 Zooko's triangle

In this section, we are introducing Zooko's triangle conjecture and our attempt to square it.

Zooko's triangle says that out of these three properties [**?**], you can usually take only two.

- *Secure*[3]: The quality that there is one, unique and specific entity to which the name maps.



- *Global*: The lack of a centralized authority for determining the meaning of a name. Instead, measures such as a Web of trust are used.
- *Memorable*: The quality of meaningfulness and memorability to the users of the naming system.

We can thus see that the systems proposed so far only gather two out of three of those properties. If we take the DNS(Sec) system with DANE extensions, we can have a memorable address that is "secure" but unfortunately doesn't have the global property because the ICANN is a centralized authority. Alternatively, Tor's Onion addresses do have the "secure" property and are global but a 16 character pseudorandom string is not memorable.

2.2 Squaring Zooko's Triangle

In the following section we are introducing a naming system that is an attempt at squaring Zooko's triangle.

Back in January 2011, Aaron Swartz described on his blog[**?**] about how Bitcoin blockchain could help in squaring Zooko's triangle. A few months later, a first implementation of that idea came into existence, Namecoin.

2.2.1 The Bitcoin Blockchain

The blockchain is Bitcoin's main innovation. Blockchains are mainly linear data-structures that were invented specifically for the Bitcoin project to store the history of all past transactions but they can be applied anywhere a distributed consensus needs to be established in the presence of malicious or untrustworthy actors.

To understand how they work we will cover the basics of how Bitcoin works. Let's take an example and see what happens when Alice tries to transfer money to Bob. Every user on the network has one address. Bitcoin addresses[**?**] are generated based a public key.

Key-Hash = RIPEMD-160(SHA-256(public key))
$\text{BTC}_{\text{Address}}$ = Base58(Version +[4] Key-Hash + Checksum)

Private-public key pairs are generated[5] when making a new Bitcoin addresses

Alice must know Bob's address to send him money. Now that Alice has Bob's address, she creates a new message saying she sends a few Bitcoins to Bob and uses her private

[2] A *pin* is a (domain name, certificate fingerprint) pair

[3] We can't agree with the naming of this property given the threat model we described previously in this paper.

[4] The + sign is a string concatenation

[5] The curved used in Bitcoin is *secp256k1* which is surprisingly a NIST recommended curve.

key to generate an ECDSA signature. Once she has generated that message and signed it, she starts gossiping about her transaction on the network, her peers hear about the transaction, they verify if Alice has enough money to make the transaction and verify the signature, if the transaction looks legitimate, they start telling all of their peers. The verification can be done thanks to the blockchain data structure which is a decentralized and unique record of all the transactions. Peers that are miners, eventually hear about this transaction and add it to the transactions memory pool. This pool is a queue of transactions that are not yet merged in the blockchain. But now how can we merge transactions into the blockchain?

2.2.2 Proof of work

The concept of *proof of work* is used to merge the blockchain. It makes adding entries in the blockchain an expensive process computationally wise. Let's say Alice wants to send Bitcoins to Bob. Alice will start gossiping on the network, telling all her peers that she wants to send money to Bob. Every client, has a copy of the blockchain and can thus assess if Alice has the amount of money she wants to transfer to Bob. If she has so, gossip will spread.

Once the miners, the workers of the blockchain, learn about a valid transaction (Alice has enough money to make the transaction and her signature is correct), they will add it to their memory pool. If the transaction is valid, the miners will add this transaction in the next block they will be mining. The benefit of making it costly to validate transactions is that validation can no longer be influenced by the number of network identities someone controls, but only by the total computational power they can bring to bear on validation.

So what is mining technically?

The hard challenge that is used in Bitcoin that needs to be solved is based on the strength of cryptographic hashes, also known as one-way functions. We consider that it is hard for someone to come up with the parameters of a hash functions for a given result. The function used in Bitcoin is *SHA-256* but this hash function could be substituted by any other. Another cryptocurrency, Litecoin, chose to use the *scrypt* function.

If I want to add some blocks (list of transactions) in the blockchain, I will have to solve this problem

SHA-256("TransactionsInfo" + challengeNumber) =< target

where *transactionsInfo* is a parameter list of information about the transactions in the blocks (and some extra information like a return address for the reward). The blockchain is vulnerable to some malleability regarding some informations in the transaction but all important information (such as the amount of transaction, recipient and sender) is part of this hash. The challenge the miner has to solve âĂŞ the proof-of-work âĂŞ is to find the challenge number such that when we append the transactions infos to the challenge number and hash the combination the output hash is smaller than a certain number. We notice that this certain challenge number is established by the network and determines how hard the problem is. In Bitcoin, this number is dynamically adjusted to keep an approximate block validation time of 10 minutes.[**?**]

When someone succeeds in solving this problem, he sends his solution to the network. Nodes verify if that answer is valid, and if it is, they broadcast it to their peers. It progressively spreads across all nodes and is added to the network's blockchain.

2.2.3 Dealing with collisions

Now what happens if two nodes, from separate parts of the blockchain do succeed in solving the challenge at almost the same time. Both nodes and their peers will spread different versions of the blockchain. We say that the blockchain has *forked*. How do we solve this?

In this case, miners will start mining the next block based on the version of the blockchain they have. If they hear that another blockchain is longer than the one they were working on before, they will switch to the longer one and put the transactions in the orphan blocks (blocks that were in the previous fork) back into the memory pool if they were not merged.

We can now understand that because every node chooses to have the longest blockchain possible, it will be very hard for an attacker to spread a fake version of the blockchain because this would involve solving the challenge for every preceding block because blocks are chained and must contain the block identifier of the previous one.

Why would one mine and spend so much computational power?

Miners are rewarded for their efforts. First, when doing a transaction, I can speed up the transfer of the money buy adding a transaction fee that will go directly to the miners. Mining software is thus optimised to sort the transactions to be merged in blocks in decreasing order of transaction fee. The other reward from mining comes from the coinbase transaction, mining does generate money. At the creation of Bitcoin, this reward was set to be a 50 BTC. But for every 210,000 validated blocks (once every four years) the reward halves. This has happened just once, to date, and so the current reward for mining a block is 25 bitcoins. This halving in the rate will continue every four years until the year 2140 CE. At that point, the reward for mining will drop below 10^{-8} bitcoins per block which is a satoshi, the smallest unit of Bitcoin and the total amount of bitcoins will cease to increase.

2.2.4 From Bitcoin to Namecoin

Now that we understand how blockchains work and why they are safe data-structures, let's now see how we can use them to square Zooko's triangle.

Namecoin is a bitcoin fork that was designed as a decentralized key-value store in addition to a crypto-currency. Putting information in the blockchain does cost a certain price.

Namecoins can be spent in many ways, here are some other use cases of the Namecoin blockchain:

- Aliases: The blockchain can be used to store an easy to remember alias for a GPG/SSH key, a Bitcoin address or any other cryptographic identity.

- Timestamping: The blockchain could store information about a specific file and from a hash of that we could find matching author name, owner, etc.
- Messaging: The blockchain could be a decentralized store for long-term messages vs BitMessage.

Writing data in the blockchain does have a certain price. Registering a domain does cost the registration fee (0.01NMC that goes to nobody) plus the transaction fee (that goes to the miner who succeeds in adding the block that contains this transaction).

The cost includes a network fee and a transaction fee. The fees are denominated in Namecoins (*NC*). Initially, the network fee was 50 NC but it decreases twice every 2 months, which means that it is already less than 1 NC after a year. This design was meant to make it expensive to register domains in the first few months to avoid the issue of domain name squatting.

Let's see what a domain name value message looks like to understand how it squares Zooko's triangle.

For a key *d/*[6] *fredericjacobs*, we have

```
{
    "ip"      : "209.236.123.133",
    "tor"     : "rqblqd3balaxcb57.onion",
    "email"   : "me@fredericjacobs.com",
    "info"    : "Frederic Jacobs",
    "tls": {
        "tcp": {
            443: [[1, "30F38EDAABC67F0344DBE27018
                552F7D575946EF", 1]]
        }
    },
    "map":
    {
        "www" : { "ip": "209.236.123.133" },
    }
}
```

This would make my website accessible at the *fredericjacobs.bit* address!

A namecoin domain needs to be renewed every 36,000 blocks which at the current rate is around 200 days. Those updates are free. Hence, unlike ICANN domain names, you don't have to pay renewal fees.

2.3 Known Issues with this new model

To be able to resolve Namecoin domains, you need to have a local copy of the complete up-to-date blockchain on your machine. In a mobile world this is currently not acceptable due to performance, bandwidth and storage restrictions. An alternative could be to deploy DNSNMC servers which would work plug-and-play with the current DNS server clients. A few of them have already been deployed[**?**]. The issue being that you will have to trust a DNSNMC[**?**] server.

Another issue that still needs to be addressed is domain squatting. Because registering namecoin domains (.bit) became ridiculously cheap, a lot of domains are being squatted by people hoping to resell those domains at some point in the future. A better pricing system that prevents massive domain registration should be adopted because registration costs did decrease too quickly to be an effective counter-measure.

3. TRANSPORT SECURITY

Now that we have a good long-term identity key distribution strategy, we are going to introduce a new transport-layer security protocol that provides safer and faster encrypted connections.

3.1 MinimaLT

MinimaLT[**?**] is a new[7] that looks really promising. protocol that presents some of the interesting features we want. It was initially designed to secure network connections between computers running the secure operating system *Ethos*. In the following sections we are going to compare it with TLS/TCP.[8] MinimaLT is implemented on top of UDP to be compatible with the current Internet infrastructure and end-systems.

3.2 Changes to make in Namecoin to make it play nicely with MinimaLT

In the original MinimaLT paper, a directory service is introduced to deal with the exchange of hosts information. We will not cover that part but will rather explain how MinimaLT can be used along with the Namecoin blockchain.

The MinimaLT directories do store the following information about the hosts: IP Address, UDP port, long-term identity key[9] and ephemeral key. We will see later in this paper why all these are needed but let's first try to change the Namecoin domain JSON structure to add these new fields.

Because we still want to be able to communicate over other protocols, we'll keep the base entry but add support for MinimaLT.

[6] The d/ prefix is used to register a .bit domain

[7] Paper published in Mai 2013

[8] We are however not going to go over the whole specification of the protocol.

[9] This key is not required in our use case where a client identifies to a server but can be useful for access control in other scenarios

```
{
    "ip"      : "209.236.123.133",
    "tor"     : "rqblqd3balaxcb57.onion",
    "email"   : "me@fredericjacobs.com",
    "info"    : "Frederic Jacobs",
    "tls": {
        "tcp": {
            443: [[1, "30F38EDAABC67F0344DBE27018552F7D575946EF", 1]]
        }
    },
    "minimaLT": {
        //"ip" : "209.236.123.133" // Only required if different than the default one
        "port": 80, // can be any UDP port.
        "id_key": long-term identity key,
        "eph_key": ephemeral key used for sending encrypted data on first RTT.
    }
    "map":
    {
        "www" : { "ip": "209.236.123.133" },
    }
}
```

For the perfect forward secrecy features of MinimaLT, we want new ephemeral keys to be announced regularly. Therefore, we also define a ephemeral key update message.

```
{
    "minimaLT": {
        "eph_key": ephemeral key used for sending encrypted data on first RTT.
    }
}
```

Because the Namecoin blockchain is a key-value store, we do not need to specify what the address of our service is because it's the key of this entry. However, we do need to sign this ephemeral key update message with our Namecoin private key.

3.3 Why better than TLS/TCP?

3.3.1 *Security is the standard*

Currently, on the Internet, most HTTP connections are unencrypted and if they are, the Apache or Ngnix configuration is often badly configured in such a way that it could be attacked. MinimaLT was designed to prevent system administrators to make security mistakes. Never will you have to manually pick cipher suites or worry about OpenSSL compatibility, MinimaLT does it all for you. Not only does it encrypt all communications between two hosts but it defaults on strong ciphers[10].

3.3.2 *Encryption & PFS*

A few months ago, a previously unknown email provider called *Lavabit* got under pressure from the US government to hand over its private SSL key. The reason turned out to be that Snowden used the email service. If Lavabit turned over the SSL keys, they would compromise the privacy of their entire user base. This is due to poorly configured SSL where the ECDHE is not the default. ECDHE performs a Elliptic curve DiffieâĂŞHellman key exchange. The last E from ECDHE stands for ephemeral which means that a new key is generated at every handshake and that key will never be stored. Hence, if later the server gets compromised or the network operator is forced to hand over an encryption key this is something that he won't be able to provide. Unfortunately, most system administrators don't put the *ECDHE* ciphers on top of their ciphers list which results in the lack of Perfect Forward Secrecy.

3.3.3 *Tunnel oriented*

MinimaLT is establishing tunnels. Tunnels provide a point-to-point encrypted channel to transmit information and have the interesting property of being more resistant to traffic analysis, just like IPSec. Tunnels mean that unlike TLS, all services from the transport layer, namely authentication, encryption, congestion control and reliability, are provided for a per tunnel basis and are not repeated per connection. MinimaLT clients have one or more connections for each of the tunnels.

3.3.4 *IP Mobility*

Thanks to the structure of MinimaLT packets, the *tunnel ID* is what identifies what packet belongs to what connection and therefore, MinimaLT has complete IP mobility. Unlike TCP, the source IP address and UDP port can change without affecting the connection. A specific RPC ($nextTid_0$) exists to announce an IP address change. A change in IP address will cause a rekeying, a procedure that we describe a little further. The fact that we have IP mobility is a big advantage over TCP that currently struggles in the mobile world when switching from a WiFi connection to a 3G signal for instance. Instead of struggling with multi-path TCP specifications we should just adopt MinimaLT that solves way more than just that issue!

3.3.5 *Handshakes & Tunnel establishment*

When establishing a single TLS connection, two hosts have to go through a TCP three-way handshake and then can start the TLS handshake which requires 4 more RTT. Thankfully MinimaLT attempts to fix that and does perform, in most cases, a cryptographic handshake in *less time than unencrypted TCP*! It's also important to note that handshakes with MinimaLT are way less frequent than in TLS because of the tunnel architecture.

Usually in TLS, the client needs to get the certificate first before being able to send encrypted data but with MinimaLT+Namecoin, we already have an ephemeral key that allows us to send encrypted data to the server. Let's see how this key exchange works.

Let's say a client C wants to connect to a server S. C uses the Namecoin blockchain to get the identity and an

[10] MinimaLT uses Curve25519 for the public-key cryptography

ephemeral key of S. Now C sends his first packet to S containing his just-generated ephemeral key, a new tunnel ID and the first bits of data. This segment's body can be encrypted using the ephemeral key of the server we retrieved previously ephemeral keys will no more be used past this point because both clients can now computer a shared secret by performing a Diffie-Hellman Key exchange with the public ephemeral keys they exchanged. In the future, this shared secret will be used to perform symmetric encryption between both hosts.

One important thing to notice is that this approach does provide perfect forward secrecy thanks to the Diffie-Hellman key exchange using ephemeral keys. Of course, perfect forward secrecy requires rekeying, the process of changing keys so that previously used keys can be cleared from memory. Let's go through the rekeying process. Rekeying can be asked by the client or the server depending on their rekey interval policies or change in IPs but the client will always be the one initiating the rekey.

When a client initiates a rekey, he generates a new tunnel ID and sends it to the server as a call. Like the initial tunnel establishment procedure, the client generates a new key pair and sends his new public key along with the new tunnel ID to the server using the current encrypted tunnel. The corresponding private key of the key pair that was generated will never be used (it actually doesn't even need to be known), it is just used to make the packet look like a regular tunnel initialisation packet. Now that the new tunnel is ready, the client can send packets to the new tunnel ID. But what encryption key should be used now? MinimaLT avoids doing a DH on rekeying for performance reasons, thus, client is hashing with a cryptographically secure hash function the old symmetric encryption key to generate the next one. The client will include the temporarily generated public key inside the encrypted packet sent to the server. Once the server gets a packet on the new tunnel ID, it will perform the same hashing as the client did to compute the decryption key. Now the server can decrypt the packet and it has to verify that the key is matching the previously sent one in the ($nextTid_0$) call. This verification is required[**?**] because otherwise an active attacker could alter the key sent in the tunnel announce packet and then sent a second packet with the matching DH computer key and notice that it would fail. He could conclude that this would be a rekeying procedure and not the creation of a new tunnel.

3.3.6 DDoS protection

What we've seen so far of MinimaLT is pretty impressive but you might have legitimate concerns of how this would play out in a threat model were attackers would like to flood a system with the creation of tunnels because DH are expensive computationally-wise. One of the MinimaLT designers, Dan Bernstein, knows this problem really well because he invented the SYN cookies. In MinimaLT, puzzles are used to address the cases where servers are under load.

When a server is not experiencing any specific load, it does accept tunnel establishments requests without any questions asked but if the server is under load, it responds with a puzzle.

The puzzle requires a proof of work to be completed. The details of the proof of work used are not covered in this paper.

Benchmarking shows that even with a puzzle presented to the client, MinimaLT handshakes are faster than TLS/TCP and in most cases (without puzzles), they are faster than unencrypted TCP.[**?**]

Another way to exhaust the server's resources would be to flood him with data to decrypt, but given the fact that past the key exchange only symmetric key encryption is used which on modern system is faster than the network links that would saturate faster.

3.3.7 Congestion control

MinimaLT's tunnel headers do contain fiels for congestion control fields such as sequence and acknowledgement numbers. MinimaLT doesn't provide anything additional to what exists in TCP for this task.

3.3.8 Flow control

Every MinimaLT connection has its own receiving window size, similar to TCP.

4. MINIMALT AND ANONYMITY

MinimaLT has no anonymity network yet similar to the Tor Project. The Tor project has been investigating on using the SPEEDY protocol (basis of the HTTP2 specification) to speed up their connections between nodes. We can imagine that if MinimaLT gets mainstream adoption, the support of MinimaLT will be seriously considered.

5. CONCLUSIONS

We've seen how MinimaLT can help us speed up our encrypted connections and with the help of Namecoin, make them safer too. The model presented in this paper clearly has flaws, mainly with the scalability of the Namecoin blockchain and the issues it causes on mobile devices. Blockchains only have a few years and many optimisations are still possible. They may not be perfect but they are the first data structure we know of that allows us to square Zooko's triangle! On the other hand, MinimaLT is a sneak peak into what tomorrow's transport-layer protocols will look like. And if that's the future, we love it.

6. CONFESSION

This paper initially aimed at digging deeper into the crypto and protocol stuff but because of concurring deadlines, this in-depth analysis of the protocols is pushed back to another paper. :)

7. ACKNOWLEDGMENTS

Still to do